ECE3073 / TRC3300
Computer Systems

Program Design and Analysis: Validation and Testing

A/Prof Lindsay Kleeman

Acknowledgement

Based on

The lecture notes of Marilyn Wolf

Computers as Components, Principles of Embedded Computing System Design

Minor modifications Clive Maynard 2020

WARNING

COMMONWEALTH OF AUSTRALIA

Copyright Regulations 1969

This material has been reproduced and communicated to you by or on behalf of Monash University pursuant to Part VB of the Copyright Act 1968 (the Act).

The material in this communication may be subject to copyright under the Act. Any further reproduction or communication of this material by you may be the subject of copyright protection under the Act.

Do not remove this notice

Goals

- **Make sure software works as intended.**
  - We will concentrate on functional testing---performance testing is harder.
- **What tests are required to adequately test the program?**
  - What is “adequate”?

Basic testing procedure

- **Provide the program with inputs.**
- **Execute the program.**
- **Compare the outputs to expected results.**

Types of software testing

- **Black-box : tests are generated without knowledge of program internals.**

- **Clear-box (white-box): tests are generated from the program structure.**

Clear-box testing

- **Generate tests based on the structure of the program.**
  - Is a given block of code executed when we think it should be executed?
  - Does a variable receive the value we think it should get?

Controllability and observability

- **Controllability: must be able to cause a particular internal condition to occur.**

- **Observability : must be able to see the effects of a state from the outside.**

Example: FIR filter

- **Code:**

```
for (firout = 0.0, j =0; j < N; j++)
      firout += buff[j] * c [j];
if (firout > 100.0) firout = 100.0;
if (firout < -100.0) firout = -100.0;
```

- **Controllability: to test range checks for firout, must first load circular buffer.**
- **Observability: how do we observe values of buff, firout?**

Path-based testing

- **Clear-box testing generally tests selected program paths:**
  - control program to exercise a path;
  - observe program to determine if path was properly executed.

- **May look at whether location on path was reached (control), whether variable on path was set (data).**

Points of view

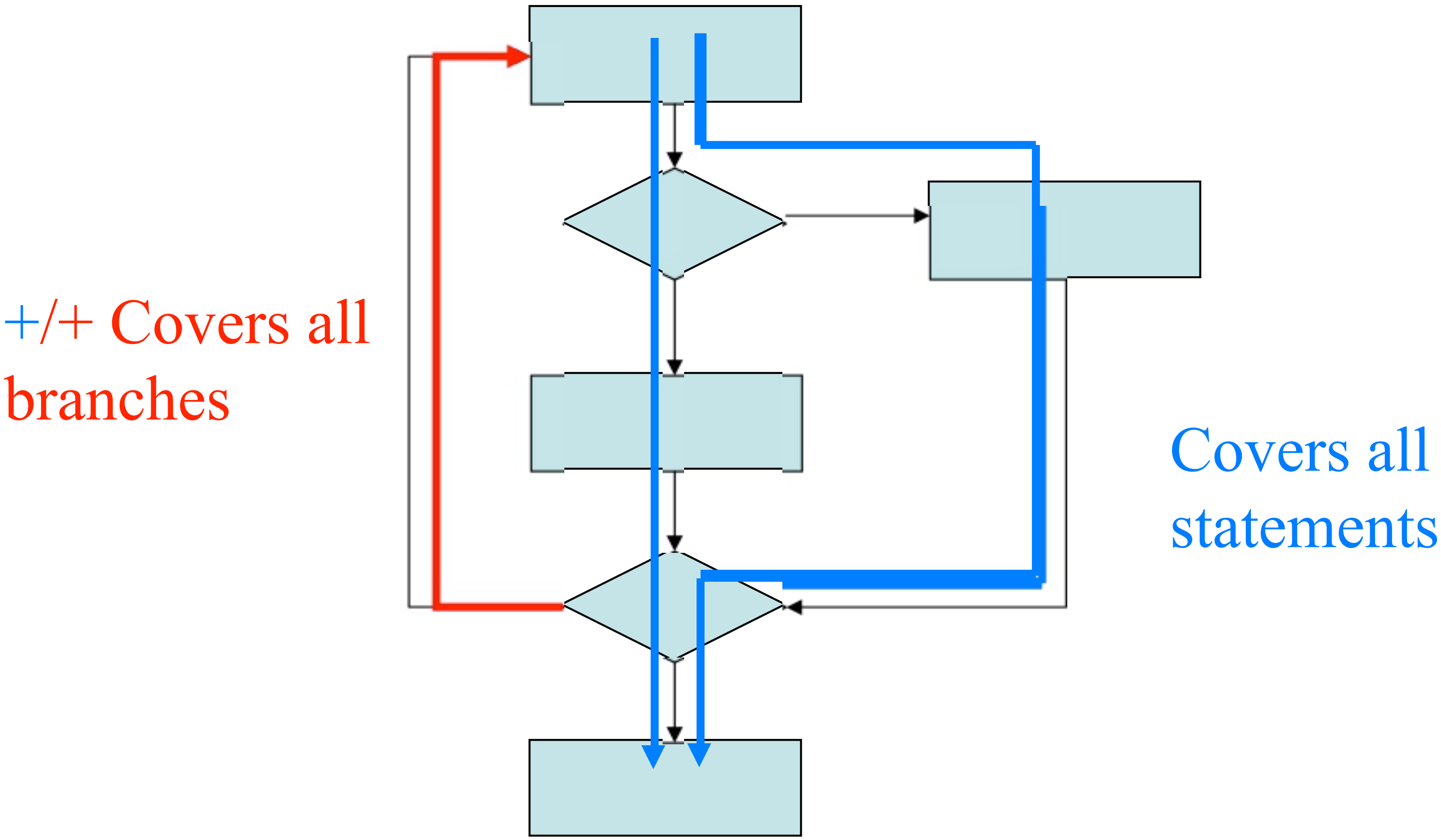
- **Several ways to look at control coverage:**
  - statement coverage;
  - basis sets;
  - cyclomatic complexity;
  - branch testing;
  - domain testing.

Example: choosing paths

- **Two possible criteria for selecting a set of paths:**
  - Execute every statement at least once.
  - Execute every direction of a branch at least once.

- **Equivalent for structured programs, but not for programs with `goto` s.**

Path example



Basis paths

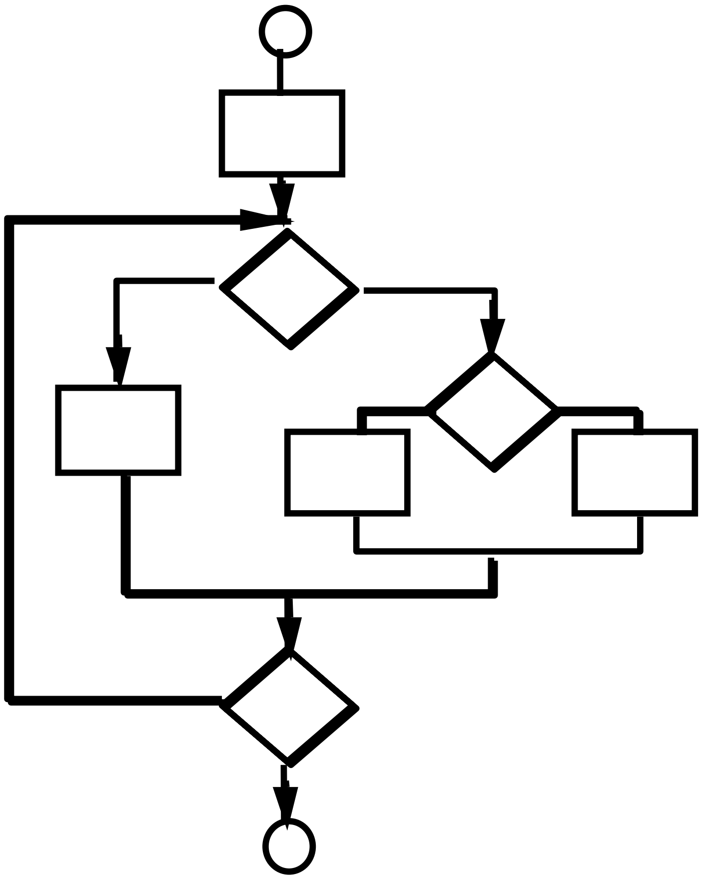
- **How many distinct paths are in a program?**
- **An undirected graph has a basis set of edges:**
  - a linear combination of basis edges (xor together sets of edges) gives any possible subset of edges in the graph.
- **CDFG is directed, but basis set is an approximation.**

Cyclomatic complexity

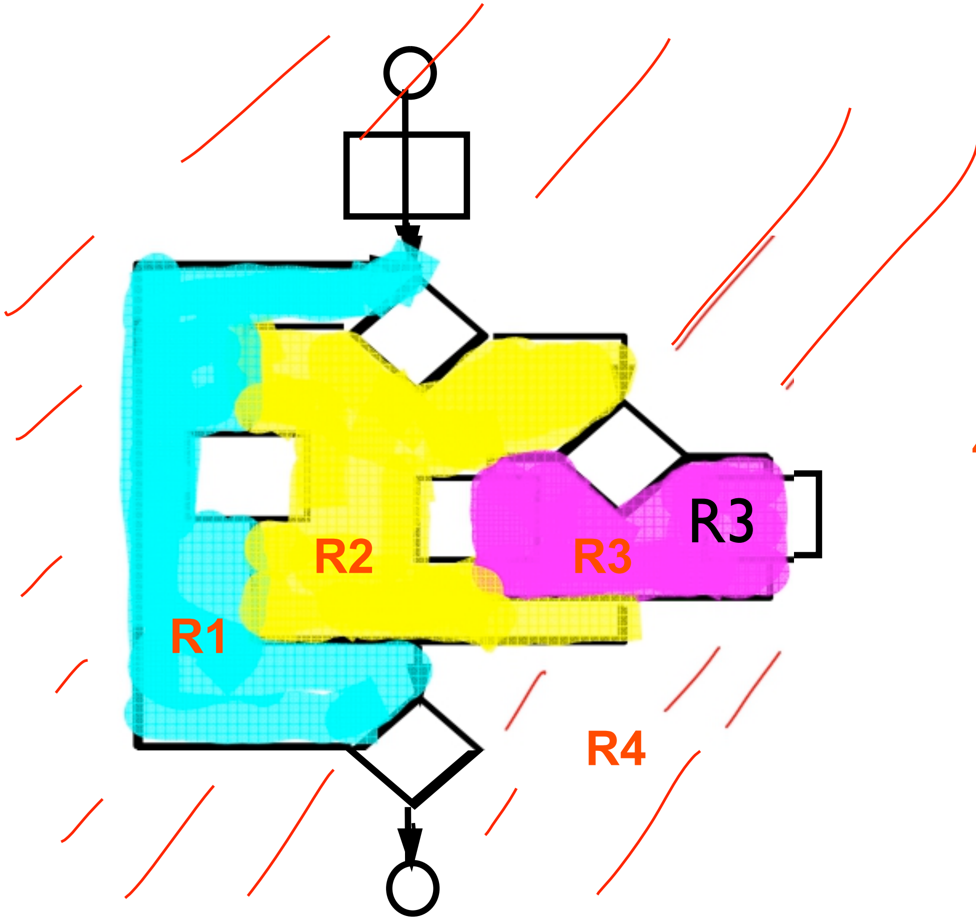
- **Provides an upper bound on the control complexity of a program:**
  - e = # edges in control graph;
  - n = # nodes in control graph;
  - p = # of components in the graph.
- **Cyclomatic complexity:**
  - M = e - n + 2p.
  - Structured program: # binary decisions + 1.
  - Also M = number of regions in graph

- **Modules with CC > 10 error prone!**

Example of Cyclomatic Complexity

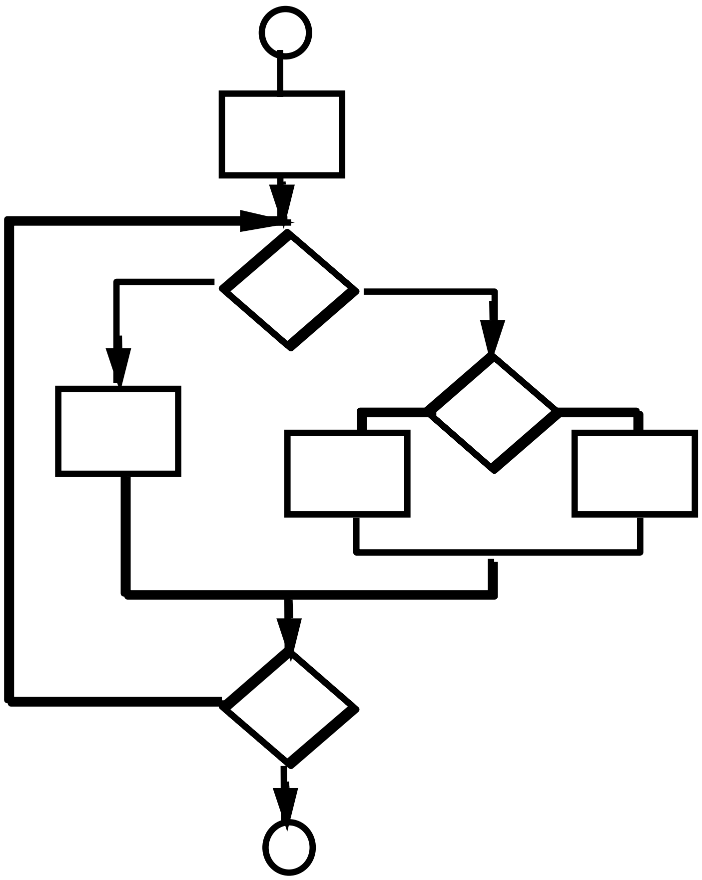


Example of Cyclomatic Complexity



4 Regions

Example of Cyclomatic Complexity

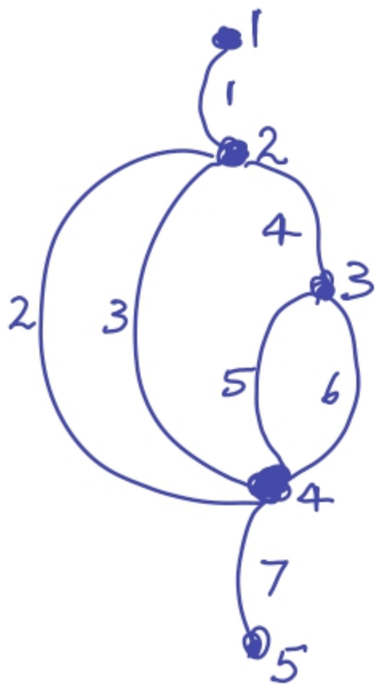


7 edges

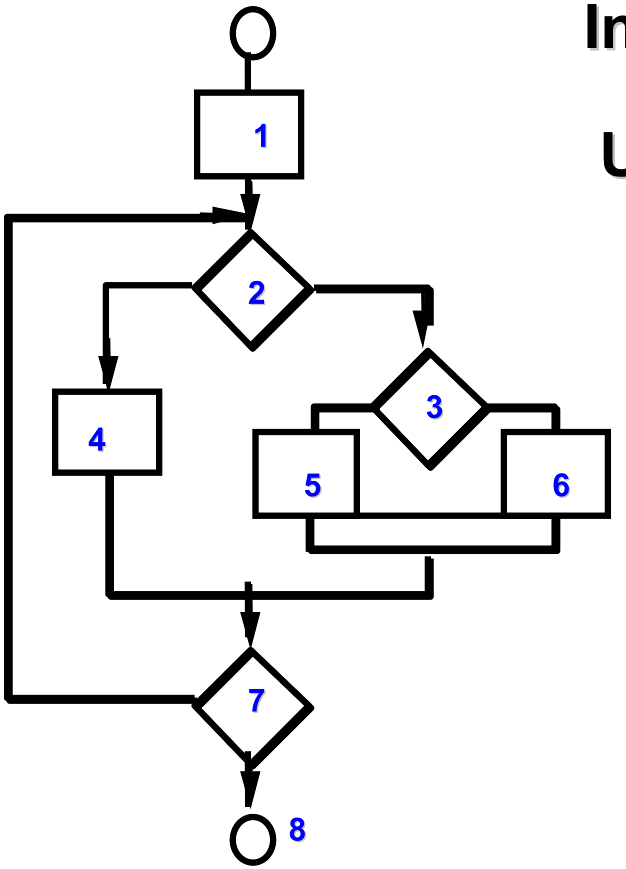
5 nodes

1 connected part

M = 7 – 5 + 2 = 4



Example of Cyclomatic Complexity



Independent paths:
M = 4,
Upper bound of 4 paths

Path 1: 1,2,3,6,7,8
Path 2: 1,2,3,5,7,8
Path 3: 1,2,4,7,8
Path 4: 1,2,4,7,2,4,...7,8

Test cases need to be generated to exercise these Paths

Branch testing strategy

- **Exercise the elements of a conditional, not just one true and one false case.**

- **Devise a test for every simple condition in a Boolean expression.**

Example: branch testing

- **Meant to write:**

  `if (a || (b >= c)) { printf("OK\n"); }`

- **Actually wrote:**

  `if (a && (b >= c)) { printf("OK\n"); }`

- **Branch testing strategy:**
  - One test is a=F, (b >= c) = T: a=0, b=3, c=2.
  - Produces different answers.

Another branch testing example

- **Meant to write:**

  if ((x == good_pointer) && (x->field1 == 3))...

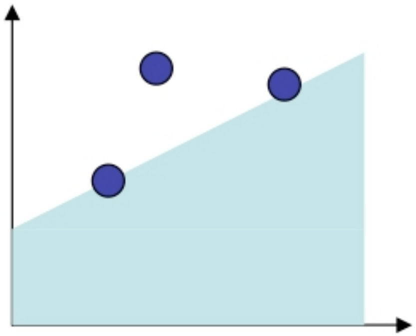
- **Actually wrote:**

  if ((x = good_pointer) && (x->field1 == 3))...

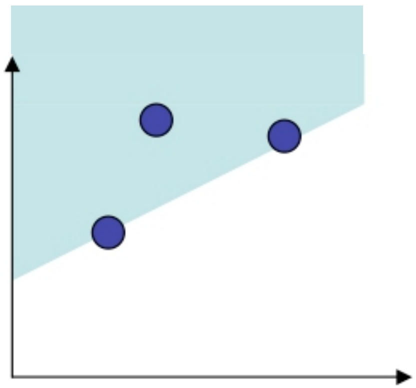
- **Branch testing strategy:**
  – If we use only field1 value to exercise branch, we may miss pointer problem.

Domain testing

- Concentrates on linear inequalities.
- Example: j <= i + 1.
- Test two cases on boundary, one outside boundary.



correct



incorrect

Data flow testing

- **Def-use analysis: match variable definitions (assignments) and uses.**
- **Example:**

x = 5;

… def

if (x > 0) ...

- **Does assignment get to the use?**

p-use

Loop testing

- **Common, specialised structure--- specialised tests can help.**
- **Useful test cases:**
  - – skip loop entirely;
  - – one iteration;
  - – two iterations;
  - – mid-range of iterations;
  - – n-1, n, n+1 iterations.

Black-box testing

- **Black-box tests are made from the specifications, not the code.**

- **Black-box testing complements clear-box.**
  – May test unusual cases better.

Types of black-box tests

- **Specified inputs/outputs:**
  - select inputs from spec, determine requried outputs.
- **Random:**
  - generate random tests, determine appropriate output.
- **Regression:**
  - tests used in previous versions of system.

Can we measure how good our testing is?

"Code reading" is a way to find a reference point for the quality of code. How many bugs were found in ostensibly completed code?

Is our testing and debugging improving our code?

Keep track of bugs found, where they were within the code and the rate of detection compare to historical trends.

Error injection : An independent person or group add bugs to a copy of the code. The developers then run their tests on the modified code and a measure of how effective that testing is derives from the percentage of these bugs found.